16156 Postel

Network Working Group
Request for Comments #505
NIC #16156

RECEIVED
JUL 16 1973

M. A. Padlipsky
MIT-Multics
6/25/73

Two Solutions to a File Transfer Access Problem

Postel

In RFC 487, Bob Bressler raises the issue of how one can use the File Transfer Protocol to send a file to a user on another system without knowing that user's password. In RFC 501, Ken Pogran points out certain objections to Bressler's solution of having a "daemon" process do the job -- in particular, the fact that it would require an interpretive access control mechanism in the daemon different from most system's normal access control mechanisms. Because Ken felt that it would be too much of a digression in RFC 501 for him to cover the following points fully, I decided it might be of interest to deal with them separately: There are at least two solutions to the problem Bob raised in RFC 487 -- in regard to "my" sending "him" a file without knowing his password -- which don't give rise to the problems noted in RFC 501. One hinges on adding a convention to the FTP, the other on adding a command.

The first solution is very straightforward. Instead of having me push the file, he could pull it. That is, he uses his own "principal identifier" (thus solving access permission problems at his end) and his own User FTP to extract the file with the aid of my Server FTP. All this requires is that 1) I give appropriate access permission at my end, and 2) he have the ability to use my Server FTP. The second condition is met by either a) his having an account on my system, or b) my system's having a known account for "free" Server FTP use. (*)

So standing the model on its head solves the functional problem, although he has to pay for the User FTP. But, then, it's he who wants the file, so why shouldn't he? On the other hand, "he" might not be logged in right now and I might be -- and by the time he can get logged in my system might be scheduled to be down. Fortunately, there's also a moderately straightforward solution to the problem as originally posed. This goes back to the mechanism used to prevent capricious and/or malicious card input on Multics: Instead of placing input (card deck or transferred file) directly into the alleged recipient's directory, place it in a "pool" directory and merely inform the recipient of its arrival. If he really wanted it, he then copies it into his own directory. That way, unauthorized people can't

---

(*) For b), I suggest that the USER NETML / PASS NETML discipline of RFC 491 be extended. That is, Hosts which allow free use of their FTP Servers should accept that pair of FTP commands as an indication to commence free service. Whether this leads to a login of a dummy user or a passoff to a daemon process is a matter of local implementation preference, of course.

Received at NIC July 5, 1973.

freeload on somebody else's directory (and the pool is, of course, periodically purged), nor can they clobber others' already-existing files.

This second solution has the virtue of requiring fewer steps than the first, and would work even when the first wouldn't; so even though it would require another FTP command, I propose the addition of a new FTP "POOL" command, which does what the last paragraph described. Depending on the various Servers' protection mechanisms, the pooled files could be made readable only by the declared recipients. This would, for example, offer an easy way to get some privacy for "mail" (which otherwise is likely to be readable by anybody who can write it), although other solutions to that particular problem of course exist. At any rate, the POOL command's syntax would be

POOL _id_ _name_

where _id_ is a valid user identifier on the Server, and _name_ is the desired name to be placed on the about-to-be-transferred file in the Server's pool directory.(*) (Servers must, of course, do whatever pre- or post-fixing to _name_ is necessary to make it unique within the pool.) The transfer then takes place in the same manner as with STOR, and on successful completion the Server sends a message to _id_ that he should pick up _name_ (suitably modified to look like a local pathname) if he wants it. The message should also identify the putative sender (even though it might have come in from a free account). The _id_ should, naturally, be validated before starting the transfer.

The question has been raised locally as to why we don't simply take a pooled view of STOR on Multics and forget about pushing for a new command. To do so would have two drawbacks, I feel: First, I think we'd be remiss in our duty as NWG participants if we failed to attempt to offer solutions to protocol problems to the Network community as a whole. Second, on a less pious but more practical note, if we don't know the _id_ we have to infer it from the pathname, which rules out abbreviations and forces senders to have to know too much about our internal structure. (The alternative of requiring an additional argument to the STOR is subject to the same objection. It is also subject to the objection that protocols really shouldn't be unilaterally extended. Of course, we could go to "site-specific parameters", but that's complicating life so much that the alternative of no unsolicited files seems preferable.) Therefore, I think that POOL would be worthwhile unless no other Servers have enough access control for it to be necessary anywhere but on Multics. At the very least, having the protocol specify an "access id" optional argument to STOR seems desirable.

(*) Note that this definition relieves the user of having to know the Server's pathname for the pool directory.

Input as to whether any of the other Servers has file access control abilities similar to those of Multics would be useful in clarifying whether this whole area is one which needs specific treatment at the protocol level, or merely needs internally acceptable handling at our end. In the meantime, if you're trying to send an unsolicited file to us for free, you can use the NETML mechanism with no directory qualification on the target pathname in the STOR, then MAIL the file name to the intended recipient, who will copy the file into his own diractory (from, in our syntax, >udd>CNet>anonymous). That's all pretty complicated, but it sure does go to show that higher-level protocols need to know an awful lot about the various operating systems. At any rate, comment on either Bressler's Problem, POOL, STOR, or other people's access control mechanisms would all be appreciated.